МАТЕМАТИКА И МАТЕМАТИЧЕСКО ОБРАЗОВАНИЕ, 2020
MATHEMATICS AND EDUCATION IN MATHEMATICS, 2020
Proceedings of the Forty-ninth Spring Conference
of the Union of Bulgarian Mathematicians
2020

"HOT RELEASES" IN LEARNING MANAGEMENT SYSTEMS AND LEARNING CONTENT MANAGEMENT SYSTEMS [*]

Oleg Iliev

One of the major problems today's web-based systems are facing is the inability to update them in real time without affecting their performance. In other words, the time when the web-based system administrators could afford the luxury of shutting down the system (making it inaccessible to the users) so that they could release a new version, is gone. An additional challenge for the administrators is the need to provide an opportunity for parallel work on different parts of the system by independent teams. This problem is significant for the modern learning content management systems (LCMS) and the learning management systems (LMS). Given their web-based accessibility by multiple users, it would be impossible to determine an interval at which the system would not be used, so as to upgrade it. Furthermore, these systems are increasingly combining a number of features with completely independent business logic, which predisposes the linking of different teams to work on different parts of the system. At the same time, it is important for these teams to be completely independent and the system to be parallel release capable. This article introduces an original model for building and releasing new system versions that enriches the component-based software architectures.

1. Introduction. Modern LMS and LCMS provide ways to access huge amounts of data for multiple users, often at the same time. Usually they are web based, located somewhere in the "cloud", and are accessed internationally. There are two important issues to be faced by the professionals who develop and maintain them: 1. "How can we ensure the system's reliability, but at the same time, to release new versions as quickly as possible, following the principles of "continuous delivery" (CD) and "continuous integration (CI)?"; 2. "How can we increase the development performance by integrating different teams to deal with different parts of the business logic of systems?". The answers to these questions often involve the use of "load balancers" and some kind of service oriented architecture. However, these approaches require either the simultaneous work of a number of specialists during the release, which dramatically slows down the process, or results in overhead in the maintenance and development. This article presents the

[*]**ACM Classification:** D.2.11.

Key words: Learning Management System, Learning Content Management Systems, Component-Based Architecture.

This research is supported through the ICT in SES program of the Ministry of Education and Science of the Republic of Bulgaria.

so called *component-oriented architecture* in combination with a caching method that introduces the concept of "hot releases". Apart from reviewing the main advantages and drawbacks of this new approach for releasing a new version, an example of an architecture that can be directly embedded in an existing LMS or LCMS is provided.

2. "Cold" vs. "Hot" release in LMS/LCMS. The release process of a web based software goes through a number of steps. Most often, they involve the following: initially the new feature or defect is developed locally by the developer; then the change is tested in an isolated environment using a test database; lastly, a test on the production environment using the live database is performed. Ignoring the need to disrupt the system in order to release the new version, this process runs the risk of delivering features to the production environment that may have an unexpected effect on the system. For example, in an LMS, due to the use of a live database that contains a different configuration for the system, the result may be different from the one seen on the test environment. For this reason, web administrators have developed a process involving the use of the so-called "canary servers" [1]. These are servers that are actually used in a production environment, but are configured to be inaccessible by the users at the time of the release. Much like the canaries were used to detect the presence of gas from the miners in order to provide timely warning of the problem, these servers are intended to provide the ability to test on a real production environment. However, this environment cannot affect the users experience. The fact is that the modern web-based systems provide simultaneous operation of multiple instances of the same system through the so-called "load balancers". This provides the ability to extract part of the production servers from the "load balancer", making it possible to test a "canary server" [2]. Good practices show that it makes sense to split the servers into two – a group to be tested first, then to return the group to the "load balancer" from which to remove the second group. Only after testing on both groups of servers the release could be considered successful. This process may be called a "cold release".

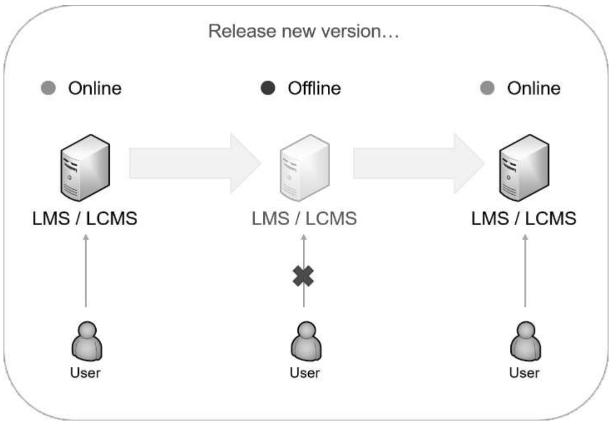

Fig. 1. A release process without a load balancer

The introduction of a "load balancer" in the architecture of LMS and LCMS can eliminate the need to interrupt the operation of the system in order to update its version.

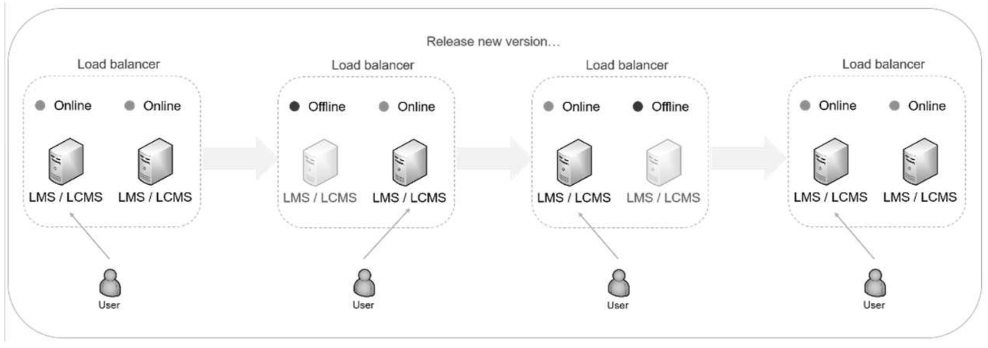

Fig. 2. A release process with a load balancer – "cold release"

The process outlined above, referred to in this article as a "cold release", is well known by the web specialists. It overcomes the system shutdown issues with a view to updating them that have been encountered until recently, but given the frequent availability of monolithic system architecture, releases remain a difficult and painstaking process. Even the slightest change to the system requires a new upgrade of the entire system. For example, changing the header used on each page would require a complete system release. Moreover, if more than one team is working on the system, their work must be well synchronized.

A process that allows the release of LCMS or LMS without necessitating the separation of system instance servers into two parts on "load balancer" level would provide additional flexibility for administrators. System administrators will no longer need to participate in the release process to manage the load balancers, enabling or disabling the public access to servers. This will already be a matter of configurations that can be made by a product manager, not by a person with strong technical background. Moreover, this process can be further developed by introducing "caching" on a HTTP level [3], which will allow the servers to restart in order to refresh their versions, but the content will still remain accessible by the users. This may be called a "hot release" of the system.

"Hot release" provides scalability and reliability of LMS and LCMS while providing the ability to work concurrently on different parts of a system. The backbone of the "hot release" is the so-called "component-based software engineering", as well as the existence of a mechanism for pre-compiling and caching the web pages served to the user. In addition to the highlighted advantages, the architecture proposed in this article can dramatically improve the performance of modern LMS and LCMS, thanks to the pre-compilation and then caching.

3. Component-based software architecture. Component-based software engineering, also called *component-based development*, is a branch of software engineering that defines the separation of problems with respect to the wide-ranging functionality available in a software system. This is based on an approach that allows the use of program logic, part of loosely coupled independent components. This approach is used to build a "component-based software architecture" [4].

Each constituent part of both LMS and LCMS can be considered a separate component. The level of granularity to be used depends on the software architect. For example, the exam section in LMS can be a separate component, the student grading section is another component, the page header and footer would also be separate components. These components can be developed and maintained absolutely independently by multiple teams – in parallel.

Once the individual components have been developed, they must be “assembled” together. This logic is defined in another component – the main component of the system or, as it is called in this paper, tha “essence” that defines what components we expect to be loaded. Finally, the actual “assembly” process is handled by the so-called “bootstrapper”, which processes the “essence” of the system where it may find the definition of the individual components. From there on the “bootstrapper” collects the pre-compiled components of which the system is composed and loads them into the solution without putting any additional logic. All business logic remains within the components themselves. The option of having different versions of the components is added to this architecture. Such an option is needed, for example, when we have a “student assessment” component that can have more than one version, because it was updated multiple times during the development process. Hypothetically, the first version may be different from the next ones by the absence of a timer that measures the time required for the student to complete the assessment.

The bootstrapper must take the appropriate version of each component as defined in the system configuration and only then assemble the components. This configuration that specifies which version of each component will be used to assemble the system is called a “manifest” in this article.

The product manager, but not the systems administrator, will be able to manage the manifests used by the system. By managing the manifests he will determine which version of the components to be used.

“How will we be able to make the release without not restarting the system and thus make it inaccessible for a certain amount of time?” is probably the most natural question here, because in order to reload the versions of the components the boostrapper requires to restart the system. In order to overcome this problem, the implementation of an HTTP level cache mechanism is provided in the architecture of the LMS or LCMS. In other words, every page that is called at least once is cached. Just the components with dynamically generated content are not fully cached. These are the components holding the data of the individual user. An example of components that are appropriate to be cached are: a “lesson” component, a “header” component.

In order to optimize the performance of LMS or LCMS, it is appropriate to automatically cache all reusable components. In other words, we do not need to wait for the user to request a component and then cache it, but we can do it upfront.

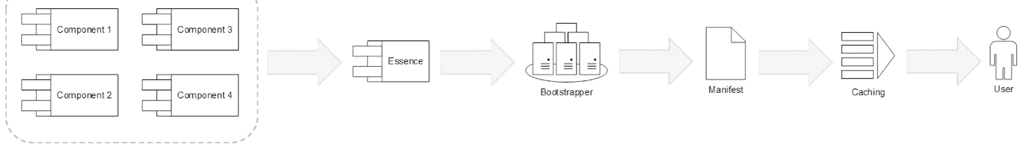

Fig. 3. Component-Based Software Architecture

The diagram (Fig. 3) describes a real architecture implemented in a prototype system for managing both learning content and learning process. It includes a number of small components, such as the header, footer, landing page, list of study materials, and more. In addition to the architecture is added one main component – “Essence”. As explained above, the components are assembled together using the bootstrapper, and finally the contents are cached. Thus, in addition to the flexibility and independence of the individual components, we can provide much higher performance.

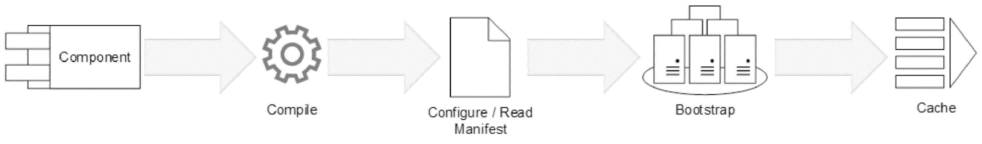

Fig. 4. Component-Based Software Architecture: Implementation Process

Figure 4 describes the process of creating a component, how to use it as part of the system, and finally, the process of caching. The component is first developed, then it is compiled, then it is added to a manifest, which is read by the bootstrapper, and finally, the assembly is cached on HTTP level.

4. Components vs. Micro-services. The architecture presented in the article outlines the question, “Since it is important to have a weak link between individual components that implement independent business logic to be able to make “hot releases” and also provide an easy way to cache and increase the performance, why don’t we go directly to using a micro-service architecture?” Micro-service architecture is the closest in ideology to component-based software architecture [5]. The main difference is that micro-services are separate server instances, operating independently of each other, having their own database and communicating with each other only through HTTP requests. In contrast, the components are parts of a rather “monolith” or service-oriented architecture (SOA) that do not always use their own databases and are being accessed internally, as part of a whole. We can apply component-oriented architecture directly to SOA, but for micro-services that is not necessary, because the full autonomy has already been achieved for them.

At first glance, micro-services provide the same flexibility, reliability, and scalability that we can provide with component-oriented architecture and even their architecture and implementation is often superior to component-oriented architecture. However, there are a few things to keep in mind:

1. The software architecture of LMS or LCMS based only on micro-services is not typical, even the largest learning management and learning content management systems do not offer such a solution. The rule usually followed when building a software solution is to go from monolithic architecture, through SOA, and then only export parts of business logic to micro-services
2. Micro-services are relatively expensive to administrate and maintain. They have separate databases and are presented through completely autonomous software instances, the responsibility for maintaining them having been dramatically increased. Instead of maintaining one primary database and one instance of the software application that can be multiplied by the use of load balancer, multiple

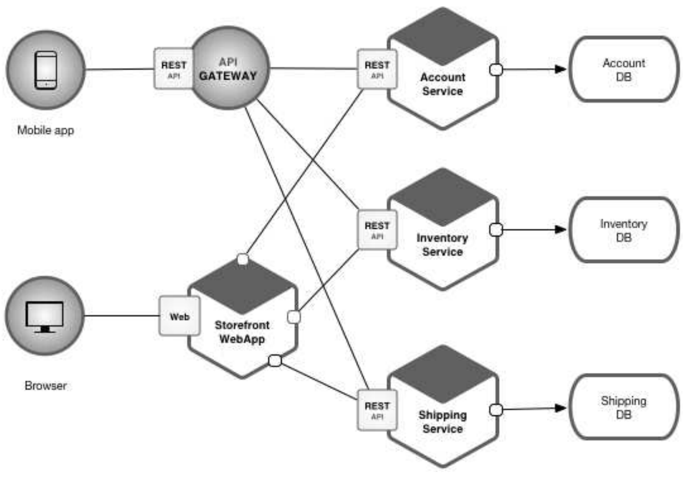

Fig. 5. Micro-service architecture

databases, multiple servers and network connections between them must be maintained.

5. Conclusion. Each web based system reaches a stage of its evolution, when the time for releasing new version that requires a downtime of the system becomes critical. Usually this is directly proportional to the number of users who use it and the number of locations from where the system can be accessed. For example, a system that has 5,000 users using it from 3 continents has no suitable time for a downtime, and in extreme cases, the downtime should be limited to a few minutes. LMS and LCMS are no an exception of this statement. Moreover, during their evolution LMS and LCMS suggest the simultaneous work of several independent teams participating in the development process. In this case, administration specialists offer "cold release" as the primary solution, followed by the migration to micro-service architecture. However, this is a relatively difficult to maintain and costly process. At the same time, component-based architectures, in combination with a special caching mechanism, provide the so-called "hot release" approach that seems to be a promising solution to all problems and challenges LMS and LCMS are facing.

REFERENCES

[1] J. Sondow, T. Ariel. Progressive deployment and termination of canary instances for software analysis. United States: Netflix Inc., 2014

[2] M. Bowman-Amuah. Load balancer in environment services patterns. United States: Accenture Global Services Ltd.

[3] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, V. Jacobson. Adaptive web caching: towards a new global caching architecture. *Computer Networks and ISDN Systems*, **30** (1998), No 22–23, 2169–2177.

[4] H. Mei, F. Chen, Y. Feng, J. Yang. (2002). An architecture-based approach for component-oriented development. Proceedings 26th Annual International Computer Software and Applications, 450–455, DOI: 10.1109/CMPSAC.2002.1045042.

[5] R. Perrey, M. Lycett. (2003). Service-oriented architecture. Proceedings of Symposium on Applications and the Internet Workshops SAINT'2003, 2003, 116–119.

Oleg Iliev
Institute of Mathematics and Informatics
Bulgarian Academy of Sciences
Acad. G. Bonchev Str., Block 8
1113 Sofia, Bulgaria
e-mail: iliev.oleg@gmail.com

„ГОРЕЩО“ ОБНОВЯВАНЕ НА ВЕРСИЯТА ПРИ СИСТЕМИ ЗА УПРАВЛЕНИЕ НА УЧЕБНОТО СЪДЪРЖАНИЕ И СИСТЕМИ ЗА УПРАВЛЕНИЕ НА ОБУЧЕНИЕТО

Олег Илиев

Един от основните проблеми, пред които се изправят съвременните уеб-базирани системи, е невъзможността за тяхното обновяване в реално време, без това да се отрази на работата им. С други думи, отмина времето, когато администраторите на уеб-базирани системи можеха да си позволят „лукса“ да изключат системата (правейки я недостъпна за потребителите), за да могат да обновят версията ѝ. Към този проблем може да се добави и допълнително предизвикателство пред администраторите – осигуряване на възможност за паралелна работа по различни части на системата от независими екипи. Този проблем е силно застъпен в модерните системи за управление на учебно съдържание (СУУС) и системите за управление на обучението (СУО). Предвид тяхната уеб-базираност и възможността за достъп на множество потребители едновременно от различни точки би било невъзможно да се определи интервал, през който системата няма да бъде използвана с цел да се обнови. Нещо повече, този тип системи все по-често комбинират редица функционалности с абсолютно независима бизнес логика, която предразполага обвързването на различни екипи за работа по отделните части на системата. В същото време е важно тези екипи да са напълно независими, без да се налага един екип да изчаква промените на друг, за да може да се направи едновременно обновяване на системата с по-малко нейни прекъсвания. В тази статия е представен оригинален модел за създаване и обновяване на системните версии, надграждащ компонентно-базираните софтуерни архитектури.